FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BarcelonaTech

Usando técnicas de visión por computador para la validación de firmas manuscritas

Trabajo Final Máster - Modalidad B

Master en Enginyeria Informàtica (MEI)
4 Julio 2019

Autora: Amanda Ceballo Iñigo
Director: Eric Borland Acosta, Sopra Steria España
Ponente: Josep Solé Pareta, Departament d'Arquitectura de Computadors (UPC)

Especial agradecimiento a Jesús Seijas por su inestimable ayuda a lo largo de todo el trabajo, a Eric Borland y a Josep Solé por sus consejos y a mi familia, por su apoyo incondicional.

Resumen

En los últimos años han aparecido muchos métodos para la verificación de identidad como el reconocimiento facial o por globo ocular. No obstante, se siguen firmando millones de contratos al día con firmas manuscritas.

Teniendo en cuenta que siempre existirá la posibilidad de que una persona pueda tener acceso a la firma de otra persona, estudiarla e imitarla deliberadamente, verificar la identidad de una persona usando su firma manuscrita es un problema poco trivial. Esto dificulta enormemente la capacidad de poder clasificar firmas como genuinas o falsificadas, o responder a la pregunta que queremos intentar estudiar en profundidad en este trabajo: ¿podemos determinar si dos firmas son la misma, sin haber visto estas nunca?

Si acotamos un poco más y sólo nos centramos en las firmas estáticas (off-line), perdemos información generada dinámicamente durante el proceso de firma, por lo que se nos dificulta aún más la tarea. Si además de esto tenemos en cuenta que la misma persona no hace nunca dos firmas iguales, esto se convierte en un problema realmente complicado.

La finalidad de este trabajo es estudiar cómo las técnicas de visión por computador actuales nos pueden ayudar dentro de este ámbito. Para ello, usaremos técnicas de preprocesado de imágenes, aprendizaje automático y redes neuronales convolucionales para la extracción de características de una firma.

Resum

En els últims anys han aparegut molts mètodes per a la verificació d'identitat com el reconeixement facial o per globus ocular. No obstant això, es segueixen signant milions de contractes al dia amb signatures manuscrites.

Tenint en compte que sempre hi haurà la possibilitat que una persona pugui tenir accés a la signatura d'una altra persona, estudiar-la i imitar-la deliberadament, verificar la identitat d'una persona amb la seva signatura manuscrita és un problema poc trivial. Això dificulta enormement la capacitat de poder classificar firmes com genuïnes o falsificades,

o respondre a la pregunta que volem intentar estudiar en profunditat en aquest treball: podem determinar si dues firmes són la mateixa, sense haver vist aquestes mai?

Si acotem una mica més i només ens centrem en les firmes estàtiques (off-line), perdem informació generada dinàmicament durant el procés de signatura, de manera que se'ns dificulta encara més la tasca. Si a més d'això tenim en compte que la mateixa persona no fa mai dues firmes iguals, això es converteix en un problema realment complicat.

La finalitat d'aquest treball és estudiar com les tècniques de visió per computador actuals ens poden ajudar dins d'aquest àmbit. Per a això, farem servir tècniques de pre-processat d'imatges, aprenentatge automàtic i xarxes neuronals convolucionals per a l'extracció de característiques d'una signatura.

Abstract

In recent years, many methods for identity verification such as facial or eyeball recognition have appeared. However, millions of contracts are still signed daily with handwritten signatures.

Keeping in mind that there will always be the possibility that a person may have access to another person's signature, study it and deliberately imitate it, verifying the identity of a person using their handwritten signature is not a trivial problem. This greatly hinders the ability to classify signatures as genuine or forged, or answer the question that we want to study in depth in this work: can we determine if two signatures are the same, without having seen these ever?

If we reduce the scope a little more and we only focus on static signatures (off-line), we lose information generated dynamically during the signature process, which makes the task even more difficult. If in addition to this we take into account that the same person never makes two equal signatures, this becomes a really complicated problem.

The purpose of this work is to study how current computer vision techniques can help

us in this field. To do this, we will use image pre-processing techniques, machine learning and convolutional neural networks for signature feature extraction.

Índice general

Índice de figuras

Índice de cuadros

Índice de fragmentos de código

Glosario

AI - Artificial Intelligence

BP - Backpropagation

CNN - Convolutional Neural Network

DBN - Deep Belief Network

DT - Decission Tree

DSN - Deep Stacking Network

GRU - Gated Recurrent Units

ILSVRC - ImageNet Large Scale Visual Recognition Challenge

ML - Machine Learning

MLP - Multilayer Perceptron

LSTM - Long Short-Term Memory

ReLU - Rectified Linear Unit(s)

RNN - Recurrent Neural Network

SVM - Support Vector Machine

Capítulo 1

Introducción

1.1. Contexto

Contratos de trabajo, recibos, documentos legales, hipotecas o pólizas son ejemplos de documentos que firmamos diariamente con nuestra firma manuscrita. En una empresa de seguros, como os podéis imaginar, se llegan a firmar millones de contratos, creciendo exponencialmente la base de datos de contratos almacenados cada día. Poder verificar la firma de estos contratos puede parecer un problema trivial para el ojo humano, pero cuando esta tarea se puede llegar a repetir un número tan grande de veces, nos llegamos a plantear si existe alguna manera automática de hacerlo y cómo las técnicas actuales de visión por computador nos pueden ayudar a resolver este problema.

1.2. Formulación del problema

Imaginémonos que somos esa empresa de seguros y vivimos de la firma de acuerdos con clientes ofreciéndoles nuestros servicios. Necesitamos que los contratos que nos firmen tengan validez, y esto pasa por asegurarnos de que los contratos que tengamos de un mismo cliente tengan la misma firma (e incluso en diferentes páginas dentro de un mismo contrato). Para ello, podemos disponer o no de la firma original de este cliente en su documento nacional de identidad, o simplemente de copias de esta firma en cada una de las hojas de los contratos. Entonces, el problema que formulamos en este trabajo es el siguiente: dadas dos firmas, ¿existe alguna manera de poder verificar automáticamente si

estas dos firmas son, en esencia, la misma?

1.3. Objetivos

El objetivo del proyecto será estudiar diferentes métodos de visión por computador para poder determinar si dos firmas manuscritas vistas por primera vez son iguales. Para ello, disponemos de la base de datos de firmas SigComp2009 [4] que contiene un total de 940 firmas de 100 usuarios diferentes. Estas firmas son genuinas, esto es, que no han sido falsificadas. La introducción en el estudio de firmas falsificadas se plantea para un futuro trabajo.

Mi objetivo personal con este trabajo es adentrarme un poco más en el mundo de la inteligencia artificial. Conocer los diferentes tipos de métodos que se utilizan y aplicarlos a un problema en concreto, enfrentarme con los problemas típicos con los que nos podemos llegar a encontrar y cómo solucionarlos. Personalmente, este es un sector que me interesa mucho y que creo que va a tener un gran crecimiento en los próximos años. Inteligencia artificial, aprendizaje profundo o aprendizaje automático son términos que me rodean cada día. Conocerlos un poco más en profundidad y poder hablar de ellos con mayor conocimiento me ayudará a afrontar futuros proyectos dentro de este ámbito con mayor seguridad. Este trabajo me ha parecido una buena oportunidad para hacerlo.

Capítulo 2

Planificación y presupuesto

2.1. Duración estimada del proyecto

La duración estimada del proyecto es de aproximadamente cinco meses. Se planifica empezar en Enero de 2019 y acabarlo en Mayo de 2019. Posteriormente, se escribirá la memoria del trabajo y preparará la presentación para el tribunal.

El tiempo estimado previsto para cada fase después de hacer la planificación lo podemos ver en la tabla 2.1.

2.2. Diagrama de Gantt

La figura 2.1 muestra el diagrama de Gantt de la planificación del trabajo con el tiempo previsto para cada una de las fases del proyecto y de las actividades llevadas a cabo.

Fase	**Horas estimadas**
Fase de planificación i definición	40h
Fase de ejecución	480h
—- Estado del arte	160h
—- Propuesta de métodos	160h
—- Evaluación de métodos	160h
Creación del prototipo	240h
—- Stack tecnológico	80h
—- Creación API	80h
—- Creación sistema para testear la API	80h
Fase de cierre	160h
—- Conclusiones	80h
—- Redactado memoria	40h
—- Preparación de la presentación	40h
Total	920h

Cuadro 2.1: Tiempo estimado del proyecto

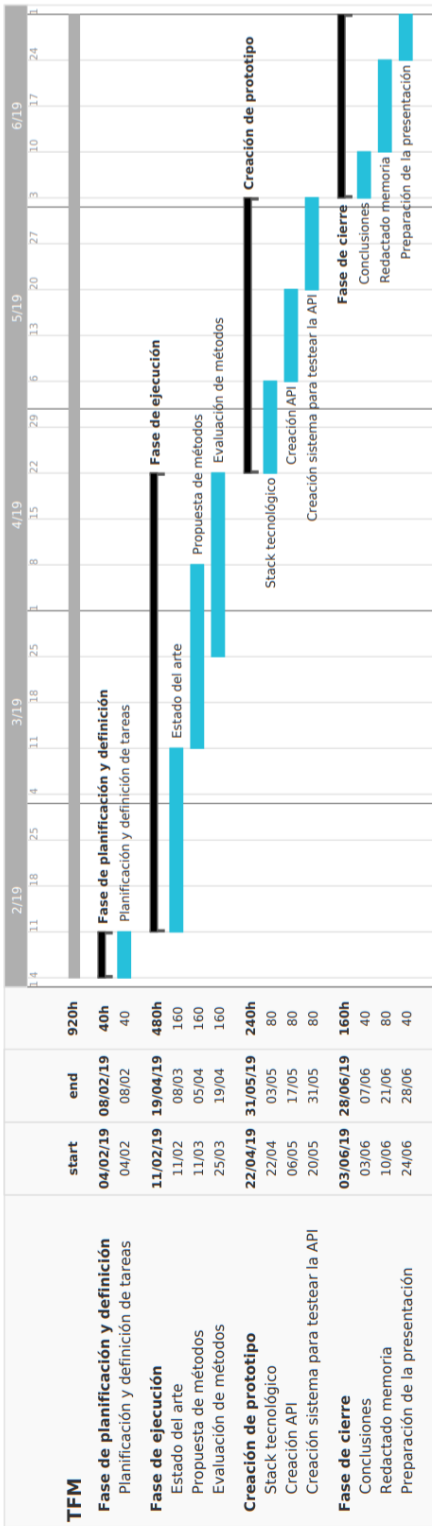

Figura 2.1: Diagrama de Gantt

2.3. Descripción de las tareas

2.3.1. Fase de planificación i definición

Durante esta fase, se planearán y definirán las tareas que se llevarán a cabo para la ejecución del trabajo. La parte primordial de esta fase es la de definir el alcance del proyecto; qué cubrirá el estudio y qué no cubrirá. De esta manera, conseguiremos centrarnos en el objetivo principal del proyecto y hacer una buena gestión de expectativas. Por otro lado, durante esta fase intentaremos asegurar que lo que vayamos a abordar se pueda entregar a tiempo para presentar el trabajo en Junio de este año. En esta fase participan activamente el director del proyecto y el ponente para asegurar que el trabajo cumple tanto las expectativas de negocio como la normativa de la universidad para el mismo.

2.3.2. Fase de ejecución

Esta es la fase más extensa del proyecto y en la que se llevarán a cabo todas las tareas necesarias para la elaboración del modelo que propondremos. Esta fase empezará con una lectura extensa del estado del arte y posteriormente se empezarán a plantear métodos que nos ayuden a resolver el problema presentado. A continuación, definiremos cómo evaluaremos estos métodos y seguiremos, paralelamente, investigando nuevos métodos que nos ayuden a mejorar cada vez más el rendimiento del modelo. La idea es, entonces, presentar un modelo básico y, poco a poco, intentar mejorarlo. Durante esta fase participa, además del programador, el director del proyecto para asegurarse de que el trabajo tiene la calidad deseada.

2.3.3. Creación del prototipo

Para poder presentar los resultados obtenidos durante el estudio, y poder evaluar de una manera realista y visual los resultados, crearemos un prototipo que nos permita testear si dos firmas son, en esencia, la misma. Este prototipo nos servirá, además, para hacer una ronda intensa de tests y poder detectar puntos de mejora que puedan ser estudiados en un trabajo futuro. En esta fase, además, deberemos planificar el tiempo necesario para el aprendizaje del paquete tecnológico que nos ayudará a crear este sistema, ya que es la

primera vez que trabajamos con él. Durante esta fase participa, además del programador, el director del proyecto para la validación de la tecnología escogida.

2.3.4. Fase de cierre

En esta fase recogeremos las conclusiones a las que hayamos llegado después de la realización de todas las actividades llevadas a cabo explicadas anteriormente. Además, haremos el redactado de la memoria del trabajo donde recogeremos todo lo aprendido. Posteriormente, prepararemos la presentación del trabajo y que defenderemos ante el tribunal de la universidad el día 1 de julio de 2019. Durante esta fase participarán, además del programador, el directo del proyecto y el ponente. El director del proyecto, para validar el cierre del trabajo y el ponente, como soporte y validación conforma a la normativa académica del redactado de la memoria y presentación.

2.4. Presupuesto del proyecto

En esta sección se recoge el presupuesto del proyecto que se ha llevado a cabo de acuerdo a la planificación de la sección anterior.

2.4.1. Recursos humanos

En la tabla 2.2 se detallan los costes relativos al personal que ha participado en el desarrollo del presente proyecto. Se consideran los siguientes participantes:
- Desarrollador del proyecto (ingeniero junior),
- Director del proyecto (ingeniero senior, experto en AI),
- Ponente (catedrático de la universidad)

La dedicación de horas aproximadas por parte del director del proyecto es de 2h/semana. La dedicación aproximada del ponente es de aproximadamente 2h/mes.

Categoría	N. Personas	Horas	Precio unitario	Total
Desarrollador	1	920	20,00€	18.400,00 €
Director de proyecto	1	42	40,00€	1,680,00 €
Ponente	1	10	60,00€	600,00 €
Total				20.680,00 €

Cuadro 2.2: Presupuesto para recursos humanos

2.4.2. Recursos para el desarrollo

La Tabla 2.3 muestra la amortización del inmovilizado material utilizado para llevar a cabo el desarrollo del proyecto. Se considera que los equipos utilizados para la realización del proyecto son amortizables en cuatro años por lo que el porcentaje aplicable a medio año es de un 12.5 %. El resto del software que se ha utilizado para el desarrollo del proyecto es de programario libre.

Descripción	Unidades	Precio Unitario	Amortización	Total amortizado
Ordenador	1	1.400,00 €	12.5 %	350,00 €
Windows 10 Pro	1	259,00 €	12.5 %	32,38 €
Total				382,38 €

Cuadro 2.3: Presupuesto de los recursos para el desarrollo

2.4.3. Resumen del presupuesto

En la tabla 2.4 se muestra el presupuesto total del proyecto incluyendo el IVA.

Descripción	Importe total
Recursos humanos	20.680,00 €
Recursos para el desarrollo	382,38 €
Subtotal	21.062,38 €
IVA	4.423,10 €
Total	25.485,48 €

Cuadro 2.4: Resumen del presupuesto total del proyecto

Capítulo 3

Estado del arte

3.1. Qué es la inteligencia artificial

Nuestra inteligencia es por lo que somos llamados *Homo sapiens*. Desde hace muchos años tratamos de entender cómo pensamos; cómo percibimos, entendemos o intentamos predecir un mundo que cada día parece más complicado. El campo de la inteligencia artificial, o IA, va mucho más allá: además de querer entender cómo pensamos, pretende construir entidades inteligentes.

IA es uno de los campos más nuevos en la ingeniería y la ciencia apareciendo su nombre como tal en 1956. Algunas de las definiciones de inteligencia artificial son recogidas en la figura 3.1.

Las definiciones de la primera fila se enfocan al razonamiento, mientras que las de la fila de abajo lo harían al comportamiento. A su vez, las definiciones de la izquierda miden el éxito en cuestiones de verosimilitud a la raza humana, mientras que las de la derecha se medirían en referencia a un rendimiento ideal, o lo que se conoce como racionalidad. Todas ellas, conformarían lo que entendemos como Inteligencia Artificial.

Pensando humanamente	**Pensando racionalmente**
"The exciting new effort to make computer think... machines with minds, in the full and literal sense" (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning..." (Bellman, 1978)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)
Actuando humanamente	**Actuando racionalmente**
"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computer do things at which, at the moment, people are better." (Rich and Kinght, 1991)	"Computer Intelligence is the study of the design of intelligent agents." (Poole et al., 1998) "AI... is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

Figura 3.1: Definiciones de inteligencia artificial

3.2. Visión por computador

Dentro del campo de la Inteligencia Artificial, la visión por computador es el conjunto de herramientas y técnicas que nos permiten obtener, procesar y analizar imágenes del mundo real con la finalidad de que puedan ser tratadas por un computador.

3.3. Aprendizaje profundo y redes neuronales convolucionales

Aprendizaje profundo, o *deep learning* en inglés, es una técnica dentro del aprendizaje automático basado en arquitecturas de redes neuronales. Está relacionado con algoritmos inspirados en la estructura y función del cerebro, constituidas como el cerebro humano: nodos de neuronas conectados como una red. Este tipo de redes están organizadas en capas, comprendiendo cada una de éstas un cierto número de nodos. Cada nodo realiza alguna operación matemática con una entrada para calcular una salida. La entrada a cualquier nodo dado es una suma ponderada de las salidas de la capa anterior (más un bias generalmente igual a uno o cero). Son estos pesos los que aprende el algoritmo

durante el entrenamiento. Para conocer estos parámetros, la salida de un entrenamiento se compara con el valor real, y el error se reparte por la red para actualizar los pesos.

Existen varios tipos de redes neuronales y se aplican en una amplia variedad de escenarios, pero los más conocidos para cada uno de estos serían los que se muestran en la figura 3.2.

Arquitectura	*Aplicación*
RNN	Reconocimiento de la voz, reconocimiento de la escritura a mano
Redes LSTM/GRU	Compresión de textos de lenguajes naturales, reconocimiento de escritura a mano, reconocimiento de voz, reconocimiento de gestos, captura de imágenes
CNN	Reconocimiento de imágenes, análisis de videos, procesamiento de lenguajes naturales
DBN	Reconocimiento de imágenes, recuperación de imágenes, comprensión de lenguajes naturales, predicción de fallos.
DSN	Recuperación de información, reconocimiento continuo de la voz

Figura 3.2: Arquitecturas de redes neuronales y sus aplicaciones más comunes

Para el problema que queremos atacar (el procesamiento de imágenes), la arquitectura más usada es la tercera: Convolutional Neural Network (CNN) o Redes Neuronales Convolucionales.

3.3.1. Arquitecturas típicas de redes neuronales convolucionales

En el año 2010 ImageNet inició el LSVRC [3]. En este concurso anual se evalúan diferentes arquitecturas de redes neuronales convolucionales para la detección de objetos y la clasificación de imágenes a gran escala. Uno de los objetivos es permitir a los investigadores comparar el progreso en la detección en una variedad más amplia de objetos. Otro de los objetivos es medir el progreso de la visión por computador para la indexación de imágenes a gran escala para la recuperación y etiquetado de datos.

En la figura 3.3 vemos las arquitecturas que han ganado este concurso en los últimos años y que han servido como arquitecturas de referencia.

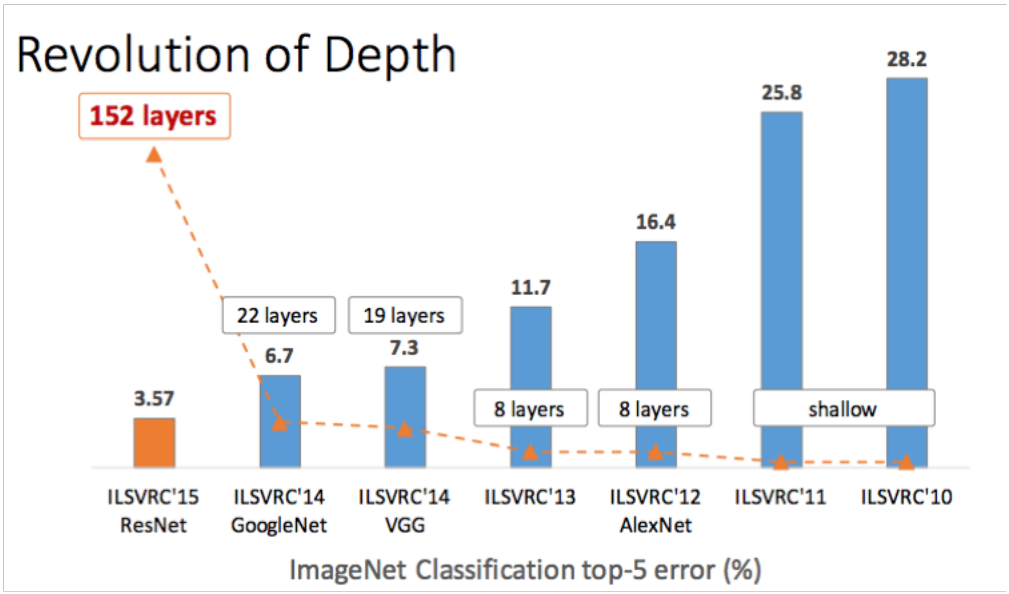

Figura 3.3: Arquitecturas ganadoras del 'Large Scale Visual Recognition Challenge' y sus tasas de error

3.3.2. Qué es una convolución

Una convolución no es más que un producto de matrices: una operación matemática donde una función se aplica a otra función. El resultado puede entenderse como una "mezcla" de las dos funciones.

¿Cómo ayuda esto a detectar objetos en una imagen? En una red convolucional, este proceso ocurre en una serie de muchas capas, cada una de las cuales lleva a cabo una convolución en la salida de la capa anterior.

Debemos entender una imagen como una matriz de bytes, ya sea rango 2 (bidimensional) o rango 3 (tridimensional, con ancho, alto y más de un canal). Entonces, una imagen en escala de grises es rango 2, mientras que una imagen RGB es rango 3 (con tres canales). Los valores de los bytes se interpretan simplemente como valores enteros, que describen la cantidad de ese canal en particular que debe usarse en el píxel correspondiente.

Para poder aplicar una convolución, se toma como origen una matriz (supongamos una imagen en escala de grises) y la convolución con una segunda matriz que conocemos como

filtro. Este proceso se repetirá a lo largo de toda la imagen y el resultado será una nueva matriz de diferentes dimensiones que la matriz de imágenes (generalmente el resultado tiene un ancho y una altura más pequeños, pero más canales). Para ilustrar este proceso, usaremos la matriz de la figura 3.4 para extraer información de la imagen.

$$\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

Figura 3.4: Matriz 3x3

En las figuras 3.5 y 3.6 se muestran las imágenes antes y después de aplicar el filtro 3x3. Como podemos ver, con este filtro lo que conseguiríamos sería 'aprender' las características verticales de una imagen.

Figura 3.5: Imagen original antes de aplicar el filtro

3.3.3. Agrupación y capas completamente conectadas

Las redes convolucionales reales rara vez se construyen solo a partir de capas convolucionales. Por lo general, también tienen otros tipos de capas. La más simple es la capa totalmente conectada o *fully connected layer*. Esta es solo una capa de red neuronal normal donde todas las salidas de la capa anterior están conectadas a todos los nodos en la

Figura 3.6: Imagen resultante después de aplicar el filtro

siguiente capa. Normalmente, estas capas aparecen hacia el final de la red.

El otro tipo clave de capa que se ven en las redes neuronales convolucionales es la capa de agrupación o *pooling*. La más común es la agrupación máxima o *max pooling*, en la cual la matriz de entrada se divide en segmentos de igual tamaño y el valor máximo en cada segmento se toma para llenar el elemento correspondiente de la matriz de salida.

$$\begin{bmatrix} 17 & 9 & 3 & 5 \\ 6 & 20 & 4 & 7 \\ 9 & 0 & 61 & 5 \\ 10 & 10 & 2 & 8 \end{bmatrix} \rightarrow \begin{bmatrix} 20 & 7 \\ 10 & 61 \end{bmatrix}$$

Figura 3.7: Ejemplo de agrupación máxima: el input es seccionado en cuadrantes de 2x2 y se selecciona el máximo como representante del cuadrante para la matriz de salida

Lo que hace este proceso es seleccionar los sectores en los que se encuentra una característica. Si nuestra red estuviera detectando caras, podríamos interpretar que existe una gran posibilidad de que haya una cara en la parte inferior derecha.

Capítulo 4

Desarrollo del estudio

4.1. Pre-procesado de imágenes

La visión por computadora y el procesamiento de imágenes en muchos casos suelen ir de la mano. Muchos sistemas de visión por computador se basan en algoritmos de procesamiento de imágenes y rara vez utilizan datos de imágenes en bruto. En su lugar, utilizan imágenes que son procesadas por un procesador de señal de imagen. Mediante algoritmos de procesamiento de imágenes se pueden transformar las imágenes de muchas maneras: suavizar, enfocar, cambiar el brillo y el contraste, resaltar los bordes, etc.

Los algoritmos de pre-procesado de imágenes utilizados en este trabajo son los mismos utilizados en estudios similares en este campo por lo que no ha formado parte del estudio y se resumen a continuación.

Las redes neuronales esperan entradas de un tamaño fijo, donde las firmas varían significativamente en forma. Primero, se centran las firmas en un gran lienzo utilizando el centro de masa de las firmas. Se elimina el fondo utilizando el algoritmo de OTSU [31] configurando los píxeles de fondo a blanco (intensidad 255), y dejando los píxeles del primer plano en escala de grises. Luego, la imagen se invierte restando cada píxel del brillo máximo I (x,y) = 255 - I (x,y) de modo que el fondo sea de valor cero. Por último, la imagen se redimensionará al tamaño de entrada de la red (952 x 1360).

La librería de Python que hemos usado es, en nuestro caso, OpenCV.

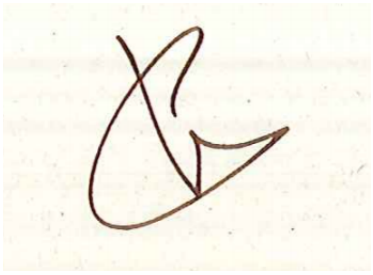

Figura 4.1: Pre-procesado de imagen. Ejemplo de imagen original.

Figura 4.2: Pre-procesado de imagen. Ejemplo de filtro de centrado.

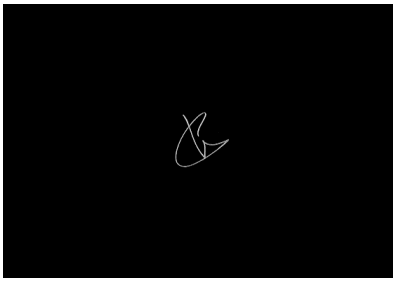

Figura 4.3: Pre-procesado de imagen. Ejemplo de imagen invertida.

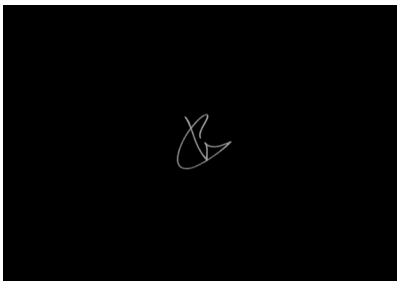

Figura 4.4: Pre-procesado de imagen. Ejemplo de imagen filtrada.

4.2. Extracción de características de firmas

El modelo utilizado para la extracción de características de firmas es el publicado en 2017 por Luiz G. Hafemanna, Robert Sabourina y Luiz S. Oliveira en el paper "Learning features for offline handwritten signature verification using deep convolutional neural networks" [1].

Mediante una red neuronal convolucional, se extrae un vector con 2048 características de una única firma. Esto son, 2048 características que definen a esta firma y que la asocian a su autor. La arquitectura usada en este estudio es una AlexNet, haciendo una extracción de características y aplicando posteriormente una capa Softmax para un clasificador. Este clasificador se aplica porque el estudio antes mencionado está enfocado a la creación de un sistema supervisado para poder clasificar firmas dentro de una base de datos de usuarios conocidos, pero el problema que abordamos en este trabajo es distinto: dadas dos firmas desconocidas, ¿podemos determinar si dos firmas son la misma? Para este trabajo, usaremos entonces hasta la penúltima capa y abarcaremos esta pregunta con el estudio de un método para la comparación de firmas, o lo que sería equivalente, de su vector de características.

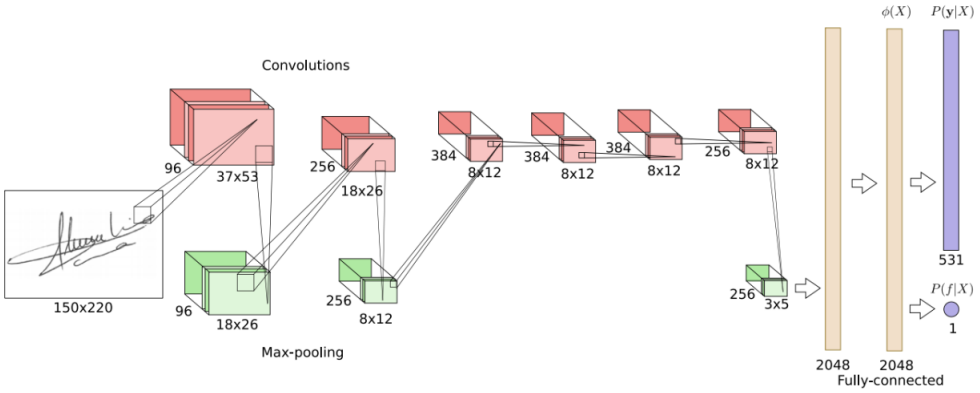

Figura 4.5: Arquitectura Alexnet.

En nuestro caso, para el proceso de extracción de firmas, generamos un fichero en formato JSON en el que, para cada una de las firmas nos guardaremos la información de la manera explicada en el fragmento de código 4.1.

```
user_features = model.get_feature_vector(processed_user_sig, layer='fc2')

str_features = ""
for f in range (0,2048):
    if f == 2047:
        str_features = str_features+str(user_features[0][f])
    else:
        str_features = str_features+str(user_features[0][f])+','

data['signatures'].append({
   'person_id': p_id,
   'feature_vector': str_features
})
```

Fragmento de código 4.1: Estructura para guardar el vector de características de todas las firmas

De esta forma, tendremos tanto el ID de la persona a la que pertenece la firma, como el vector de características extraído para esa firma en particular.

4.3. Métodos para la comparación de características

Los dividiremos en dos grandes bloques: métodos con inteligencia artificial y métodos 'sin inteligencia artificial' refiriéndonos únicamente a la comparación de características puesto que para extraerlas ya estamos haciendo siempre uso de AI.

4.3.1. Métodos sin inteligencia artificial

Para poder comparar las características de dos firmas diferentes f y f' una vez extraídas, hemos empezando planteado los métodos a continuación, teniendo en cuenta f≥0:

Dadas dos f y f' iguales, en cualquiera de los tres métodos propuestos, el sumatorio del valor de sus características será 0.

Método comparación	*Función*
Diferencia aritmética	$\sum \lvert f_i - f'_i \rvert \quad , \; 0 \geq i \geq 2047$
Diferencia de cuadrados	$\sum \lvert f_i^2 - f'^2_i \rvert \quad , \; 0 \geq i \geq 2047$
Exponencial	$\sum e^{f_i - f'_i} - 1 \quad , \; 0 \geq i \geq 2047$

Figura 4.6: Métodos estudiados para la comparación de características

Para cada comparación, además del sumatorio de firmas, nos hemos guardado la variable objetivo del sistema supervisado que queremos crear: *sameUser*, que indicará si las firmas han sido hechas por el mismo usuario o no. Esto podremos saberlo comparando el personId que nos hemos guardado anteriormente en el proceso de extracción de firmas.

```
1 if (person1_id != person2_id):
2     same_user = '0'
3 else:
4     same_user = '1'
```

Fragmento de código 4.2: Determinación de la clase objetivo

Para obtener instancias de la clase sameUser = ’0’ (firmas de usuarios diferentes) y de la clase sameUser = ’1’ (firmas del mismo usuario) hemos cruzado las firmas de manera que compararemos todas las del mismo usuario entre ellas para obtener 11 + 10 + 9 + + 1 = 66 muestras de la clase sameUser = ’1’ , y a la hora de obtener muestras de la clase sameUser = ’0’, sólo compararemos con una firma de cada usuario diferente al de origen. De esta manera obtendremos una distribución más equilibrada entre las dos clases. La diferencia al aplicar este método o aplicar el cruzado de datos completo (todas las firmas con todas) lo podemos ver en las figuras 4.7 y 4.8.

Para poder visualizar esta distribución hemos utilizado la librería matplotlib.pyplot y dibujado la variable s (sumatorio del valor de las características) para cada una de las dos clases.

```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv(<CSV_FILE>,usecols=['s','same_user'])

g1 = data[data['same_user'] == 1]
g2 = data[data['same_user'] == 0]

g1['s'].hist(alpha=0.8) #blue = same user
g2['s'].hist(alpha=0.5) #orange = diff user
```

Fragmento de código 4.3: Representación de las clases "Mismo usuario.º Üsuario diferente"según el sumatorio del valor de sus características.

En `<CSV_FILE>` usaríamos los tres ficheros .csv explicados en la sección anterior.

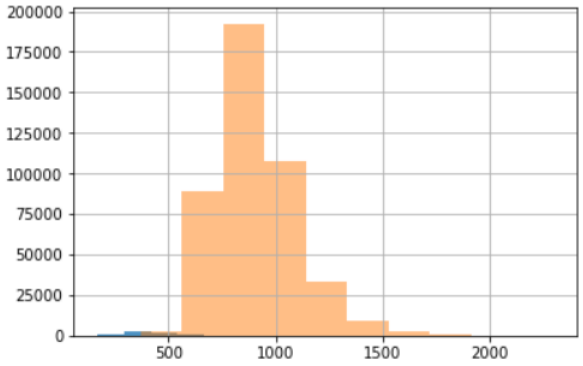

Figura 4.7: Comparación de firmas mediante método diferencia aritmética (cruzado datos completo).

Las figuras 4.9 y 4.10 muestran la distribución conseguida entre las dos clases aplicando los métodos de diferencia de cuadrados y diferencia exponencial, y el cruzado de datos simplificado. El método que pensamos que mejor funcionará será la diferencia exponencial. ¿Por qué?

Si la característica f1 de la firma del usuario 1 es 8.01 y la característica f1 del usuario 2 es 8.03, cuando apliquemos la comparación exponencial, el resultado de la operación continuará siendo del orden 0.10, de la misma manera que lo haría si aplicáramos la di-

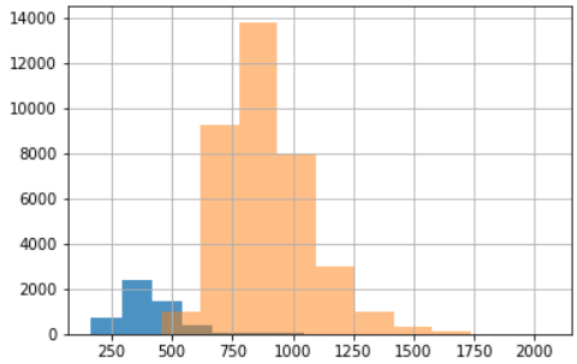

Figura 4.8: Comparación de firmas mediante método diferencia aritmética (cruzado datos simplificado).

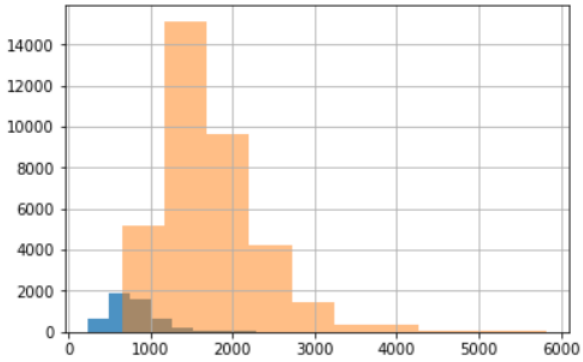

Figura 4.9: Comparación de firmas mediante diferencia de cuadrados (cruzado datos simplificado).

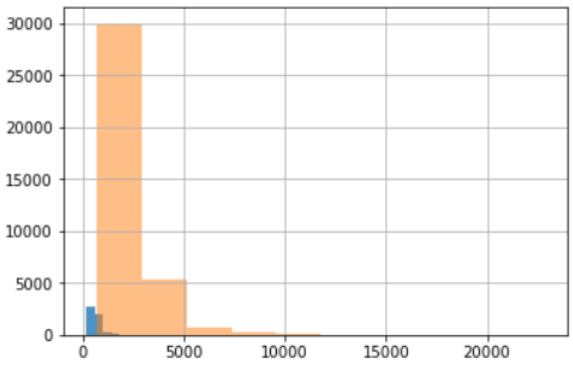

Figura 4.10: Comparación de firmas mediante diferencia exponencial (cruzado datos simplificado).

ferencia aritmética simple.

Sin embargo, si la característica f2 de la firma del usuario 1 es 9 y la característica f2 del usuario 2 es 10, cuando apliquemos la comparación exponencial, el resultado de la operación será 1.71. Si aplicáramos la diferencia aritmética simple, la diferencia sería 1. De esta manera conseguimos que dos características que no se parecen, se parezcan un poco menos y obviar aquellas pequeñas diferencias entre dos características concretas. Hay que tener en cuenta que las características que estamos comparando tienen un valor numérico con hasta trece decimales.

4.3.2. Métodos con inteligencia artificial

Hasta ahora hemos hablado de varios algoritmos muy sencillos que no incluyen ningún método de inteligencia artificial. Dado que el problema que estamos atacando es un problema de clasificación binaria típico, hemos querido probar cómo nos podía ayudar introducir inteligencia artificial o, más concretamente, aprendizaje automático. Como vemos en la figura 4.11, la clasificación de imágenes es uno de los problemas más típicos dentro del aprendizaje supervisado.

Figura 4.11: Tipos de aprendizaje automático y problemas típicos que resuelven

Dentro de los algoritmos típicos de clasificación, encontramos los siguientes:

- Naive Bayes
- Árboles de decisión
- Random Forest
- Support Vector Machine (SVM)
- Regresión logística

SVM se aplica cuando estamos tratando variables en un espacio multidimensional, por lo que no aplica a nuestro problema. La regresión logística tampoco nos ayudará demasiado ya que como hemos visto analizando nuestros datos, hay puntos donde las dos clases se solapan y son precisamente estos los que necesitamos que la inteligencia artificial nos resuelva. Nos centraremos, por tanto en los tres primeros. Los resultados los podremos ver en la sección 4.4.

4.3.3. Generalización por rangos

Una consideración importante para aprender una buena función objetivo a partir de los datos de entrenamiento es qué tan bien se generaliza el modelo a nuevos datos. La generalización es importante porque los datos que recopilamos son sólo una muestra, incompleta y ruidosa y los ejemplos que aún no ha visto nuestro modelo son infinitos.

Cuando hablamos de inducción, nos referimos al aprendizaje de conceptos generales a partir de ejemplos específicos, que es exactamente el problema que los problemas de aprendizaje automático supervisado pretenden resolver. Esto es diferente de la deducción que es al revés y busca aprender conceptos específicos de las reglas generales. La generalización se refiere a qué tan bien los conceptos aprendidos por un modelo de aprendizaje automático se aplican a ejemplos específicos que el modelo no vio cuando estaba aprendiendo.

El objetivo de un buen modelo de aprendizaje automático es generalizar bien los datos de entrenamiento a cualquier dato del dominio del problema. Esto nos permite hacer predicciones en el futuro sobre los datos que el modelo nunca ha visto. Existe una terminología utilizada en el aprendizaje automático cuando hablamos acerca de qué tan bien un modelo de aprendizaje automático aprende y generaliza los nuevos datos, es decir, el

grado de sobre ajuste (en inglés conocido como *overfitting* o *underfitting*). Ambas son las dos causas principales del bajo rendimiento de los algoritmos de aprendizaje automático.

En nuestro caso, el método que hemos usado para aplicar una generalización de los datos es el siguiente. En lugar de sumar las 2048 características de las dos firmas, vamos a crear diferentes rangos y contar las características que caen en cada rango una vez hecha la diferencia exponencial.

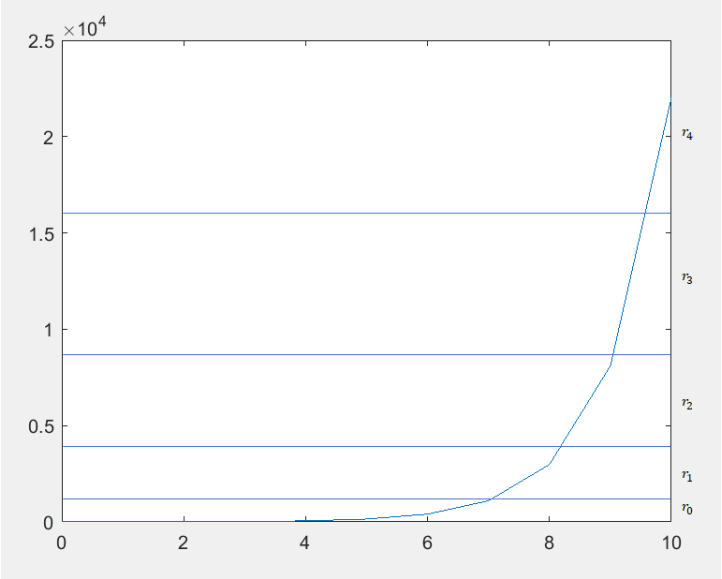

Figura 4.12: División por rangos de las características de una firma

Haremos varias pruebas para ver sobre qué valores situaremos los rangos para que nos de el mejor resultado y crearemos un sistema de aprendizaje supervisado (por ejemplo, un árbol de decisión) entrenado con el conteo de características que caen en cada rango para crear un modelo que prediga si dos firmas son la misma.

Supongamos que tenemos una estructura de datos como la que se muestra en la tabla 4.13. Mediante la implementación de la librería de sklearn mostrada en el fragmento de código 4.4, lograremos crear el modelo predictivo.

```
#Identify features and the target variable
feature_cols = ['r0','r1','r2','r3','r4','r5']
X = data[feature_cols]
y = data.same_user

#Split data to train and test
from sklearn import model_selection
X_train, X_test, y_train, y_test = model_selection.train_test_split(X,y,↩
    test_size=0.3,random_state=0)

#Train the model
dtree = tree.DecisionTreeClassifier(max_depth=5,random_state=0)
dtree = dtree.fit(X_train, y_train)

#Save the model
filename = 'finalized_model.sav'
joblib.dump(dtree, filename)
```

Fragmento de código 4.4: Determinación de la clase objetivo

	r0	r1	r2	r3	r4	r5	s	same_user
0	1419	158	240	180	47	4	613.545809	1
1	1459	166	236	154	32	1	499.918565	1
2	1510	175	238	102	23	0	410.163813	1
3	1279	116	198	210	172	73	1686.571891	1
4	1474	151	224	160	37	2	530.050409	1

Figura 4.13: Matriz de datos obtenida mediante el método de generalización (5 primeros registros)

En la figura 4.14 vemos el árbol de decisión que se crearía mediante la implementación del árbol de decisión con la generalización por rangos.

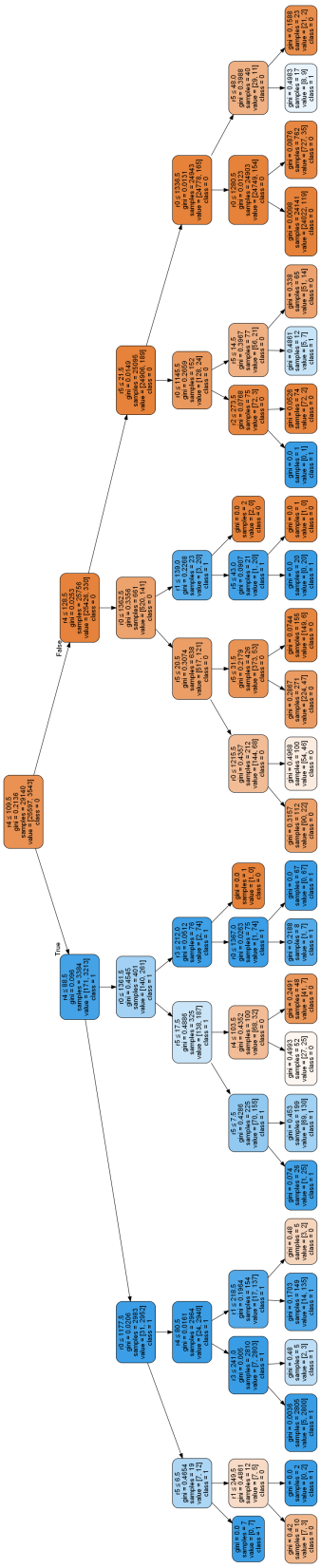

Figura 4.14: Árbol de decisión obtenido una vez aplicado el método para la generalización del modelo

4.4. Análisis e interpretación de los resultados

En el campo del aprendizaje automático y específicamente en el problema de la clasificación, una parte primordial para el estudio es la evaluación de los métodos propuestos. La manera de poder evaluar el rendimiento de un modelo es mediante la matriz de confusión mostrada en la figura 4.15. Con esta matriz se prueba para un conjunto de datos de prueba de los cuales se conoce el valor real el resultado del algoritmo y se compara con el resultado esperado.

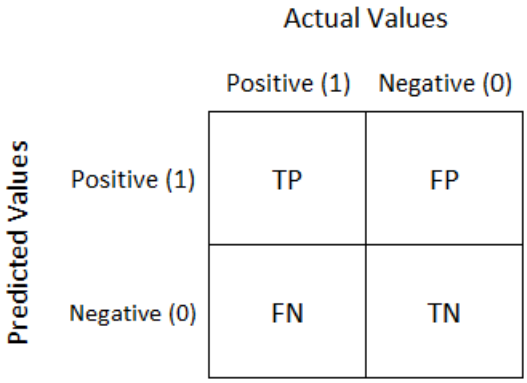

Figura 4.15: Matriz de confusión o *confusion matrix*

Verdadero Positivo (TP): Interpretación: Se predijo positivo y es cierto. Predijimos que la firma pertenecía al mismo usuario, y la firma pertenece al mismo usuario.

Verdadero Negativo (TN): Interpretación: Se predijo negativo y es cierto. Predijimos que la firma no pertenecía al mismo usuario, y la firma no pertenece al mismo usuario.

Falso Positivo (FP): (Error Tipo 1) Interpretación: Se predijo positivo y es falso. Predijimos que la firma pertenecía al mismo usuario, y la firma no pertenece al mismo usuario.

Falso Negativo (FN): (Error de Tipo 2) Interpretación: Se predijo negativo y es falso. Predijimos que la firma no pertenecía al mismo usuario, y la firma pertenece al mismo usuario.

Usando la matriz de confusión, podemos obtener las métricas de la figura 4.16.

Métrica	*Descripción*	*Fórmula*
Exactitud	En general, ¿con qué frecuencia es correcto el clasificador?	$\frac{TP+TN}{TP+TN+FP+FN}$
Error de clasificación	En general, ¿con qué frecuencia es incorrecto el clasificador?	$\frac{FP+FN}{TP+TN+FP+FN}$
Sensitividad	Cuando el valor real es positivo, ¿con qué frecuencia es correcta la predicción?	$\frac{TP}{FN+TP}$
Especificidad	Cuando el valor real es negativo, ¿con qué frecuencia es correcta la predicción?	$\frac{TN}{FP+TN}$
Tasa de falsos positivos	Cuando el valor real es negativo, ¿con qué frecuencia la predicción es incorrecta?	$\frac{FP}{FP+TN}$
Precisión	Cuando se predice un valor positivo, ¿con qué frecuencia es correcta la predicción?	$\frac{TP}{TP+FP}$
F1	Promedio ponderado de precisión y *sensitividad*	$\frac{Precisión \cdot Sensitividad}{Precisión + Sensitividad}$

Figura 4.16: Métricas para la evaluación del modelo que se pueden obtener a partir de la matriz de confusión.

Veamos las métricas que nos salen para cada uno de los modelos propuestos. Partiremos del método basado en la suma de diferencias aritméticas. Recordemos que aquí no hemos usado ningún método para la generalización del modelo pero podríamos llegar a plantear lo siguiente. Situemos un corte entre las dos clases y consideremos que todo lo que caiga a la izquierda de ese corte son firmas hechas por el mismo usuario y todo lo que caiga a la derecha del mismo, son firmas hechas por usuarios diferentes. Probando varios cortes, encontramos que el más óptimo se sitúa en el punto 550 consiguiendo esta matriz de confusión:

Verdaderos positivos (TP): 4650
Falsos positivos (FP): 230
Falsos negativos (FN): 491
Verdaderos negativos (TN): 36258

Con estos valores, y basándonos en las métricas explicadas anteriormente conseguimos los resultados que se muestran en la tabla 4.1.

Métrica	**Valor**
Exactitud	98.2680 %
Error de clasificación	1.7319 %
Sensitividad	90.4493 %
Especificidad	99.3696 %
Tasa falsos positivos	0.6303 %
Precisión	95.2868 %
F1	92.8051 %

Cuadro 4.1: Rendimiento del modelo bansándonos en la suma de diferencias aritméticas

Para el método de diferencia aritmética podemos hacer el mismo procedimiento, pero poniendo el corte en otro punto. Esta vez, el punto de corte para la suma de las diferencias de cuadrados con el que conseguimos mayor rendimiento del modelo es el 950, obteniendo las métricas de la tabla 4.2. Como podemos ver, los resultados son peores que los obtenidos con el primer modelo basado en la diferencia aritmética.

Métrica	**Valor**
Exactitud	94.6599 %
Error de clasificación	5.3400 %
Sensitividad	74.6741 %
Especificidad	97.4758 %
Tasa falsos positivos	2.5241 %
Precisión	80.6512 %
F1	77.5477 %

Cuadro 4.2: Rendimiento del modelo basándonos en la suma de diferencias de cuadrados

Aplicando el método de suma de diferencias exponenciales, obtenemos los resultados de la tabla 4.3 y vemos que el rendimiento supera el primero planteado con la diferencia aritmética, por lo que nuestra teoría de que éste era el método que mejores resultados nos iba a dar se confirma.

Veamos ahora el rendimiento qué obtenemos aplicando inteligencia artificial y el método de generalización propuesto. Cogeremos como medida de referencia la métrica F1 ya que tiene en cuenta tanto la la precisión como la sensitividad del modelo para valorar el rendimiento.

Métrica	**Valor**
Exactitud	98.3304 %
Error de clasificación	1.6695 %
Sensitividad	91.3051 %
Especificidad	99.3203 %
Tasa falsos positivos	0.6796 %
Precisión	94.9817 %
F1	93.1072 %

Cuadro 4.3: Rendimiento del modelo bansándonos en la suma de diferencias exponenciales

Algoritmo	**Impl. simple**	**Impl. por rangos**
Naive bayes	98.4238 %	98.7439 %
Árbol de decisión	97.9731 %	98.4621 %
Random forest	98.3376 %	98.9815 %

Cuadro 4.4: Métrica F1 de los modelos obtenidos aplicando inteligencia artificial para la clasificación de firmas según el valor por rangos de sus características.

El mejor resultado lo hemos obtenido con el Random Forest y aplicando el método de generalización propuesto, consiguiendo un F1 de un 98.9815 %. Esto significa que, con ese porcentaje de acierto, nuestro modelo puede predecir si dos firmas son la misma. Tal como planteábamos al principio del trabajo, en nuestro estudio no hemos añadido la detección de firmas falsificadas, aunque consideramos que este es un buen punto de partida que nos puede acercar más a ese objetivo.

Capítulo 5

Prototipo

En el siguiente capítulo, explicaremos cómo hemos creado un prototipo de sistema de predicción que determina si dos firmas son la misma.

En el diagrama 5.1 vemos una arquitectura muy básica que hemos construido para poder ver en funcionamiento el validador de firmas a probar. Se divide principalmente en dos partes: front-end y back-end. En el front-end disponemos de una interfaz que se encargará de crear dos imágenes con las dos firmas que queramos verificar. En el back-end disponemos de una API REST y del servicio de validación de firmas. En este último es donde se encuentra toda la inteligencia artificial y el modelo predictivo que hemos creado.

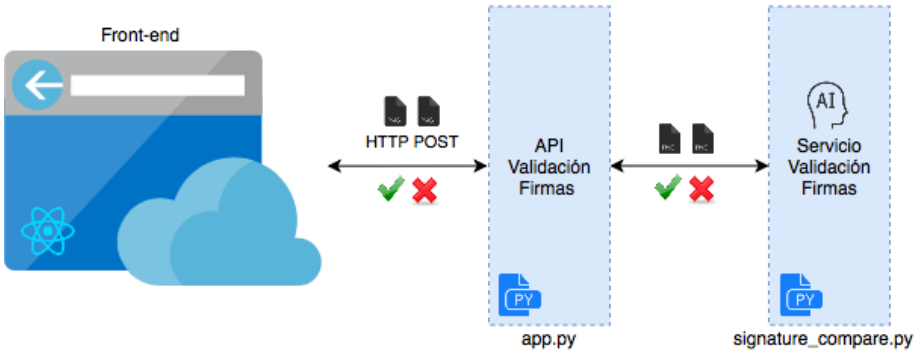

Figura 5.1: Arquitectura del sistema prototipo diseñado para la verificación de firmas manuscritas

5.1. Front-end

Para la creación del front-end hemos utilizado la biblioteca de ReactJS. Esta biblioteca JavaScript es de código abierto y fue diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Es mantenida por Facebook y la comunidad de software libre y han participado en el proyecto más de mil desarrolladores diferentes.

Partimos del complemento 'Signature Pad' citado en la fuente [16]. Este complemento nos permite capturar la firma de un usuario mediante una pantalla táctil. Duplicando este complemento, podemos conseguir crear a la vez las dos firmas que queremos validar. Con el fragmento de código HTML 5.1 creamos la interfaz del sistema. En ella tendremos los dos recuadros para capturar las firmas y dos botones: uno para hacer la llamada a la API y el otro para poder borrar lo que hayamos dibujado. Esto nos permitirá hacer múltiples pruebas en una misma sesión sin tener que reiniciar la página cada vez.

```
<section class="signature-component">
  <h1>Signature validation using Convolutional Neural Network </h1>
  <row>
    <canvas id="signature-pad" width="800"  height="400"></canvas>
    <canvas id="signature-pad2" width="800" height="400"></canvas>
  </row>
  <div>
    <button id="save">Validate</button>
    <button id="clear">Clear</button>
  </div>
</section>
```

Fragmento de código 5.1: Código HTML del front-end

La funcionalidad principal del front-end es enviar estas imágenes al back-end para validar las dos firmas, recoger el resultado que nos envíe el servicio de validación de firmas y mostrar el resultado (OK o KO) por pantalla. El servicio para validar las firmas lo hemos expuesto en una URL pública con Ngrok (esto lo veremos en el siguiente apartado). La funcionalidad descrita deberá ser llamada cada vez que hagamos clic en el botón de Validar, por lo que hemos añadido mediante ReactJS un evento nuevo a este botón que

lance el proceso cada vez que se haga clic en éste. En el fragmento de código 5.2 vemos cómo hacemos la declaración del evento. Como podemos ver, en él se crean las imágenes en formato PNG en base a lo que haya en ese momento pintado en los recuadros, se crea la llamada POST donde añade en un 'formData' las imágenes, se hace el fetch al servicio expuesto en NGROK y se recoge la respuesta que devuelve el servicio. Según si esta respuesta es 1 (las firmas son consideradas pertenecientes a una misma persona) o 0 (no pertenecen a la misma persona), colorearemos los bordes de los recuadros en verde o rojo. En las figuras 5.2 y 5.3 vemos un ejemplo de cada caso. El botón de borrado vuelve los recuadros al color original.

```
validateButton.addEventListener('click', function(event) {
  sig1 = canvas.toDataURL("image/png");
  sig2 = canvas2.toDataURL("image/png");

  var blob1 = dataURItoBlob(sig1);
  var blob2 = dataURItoBlob(sig2);

  const formData = new FormData();
  formData.append('sig1', blob1);
  formData.append('sig2', blob2);
  const options = {
    method: 'POST',
    body: formData,
    mode: 'cors'
  };

 fetch('https://ea90d2e7.ngrok.io/sig_compare', options)
    .then((resp) => resp.json())
    .then(function(data) {
      if(data.message[0]==1){
        canvas.style.borderColor = 'lightgreen';
        canvas2.style.borderColor = 'lightgreen';
      }else{
        canvas.style.borderColor = 'red';
        canvas2.style.borderColor = 'red';
      }
      canvas.style.borderWidth = '2.5px';
      canvas2.style.borderWidth = '2.5px';
    })
    .catch(function(error) {
    // If there is any error you will catch them here
    });
});
```

Fragmento de código 5.2: Llamada al servicio API REST desde el front-end

Para poder exponer el front-end en un entorno público y probarlo desde cualquier tablet con conexión a internet, hemos escogido la plataforma Codepen (`https://codepen.io`). Esta nos permite añadir código HTML, JavaScript y CSS en un proyecto y poderlo probar en tiempo real.

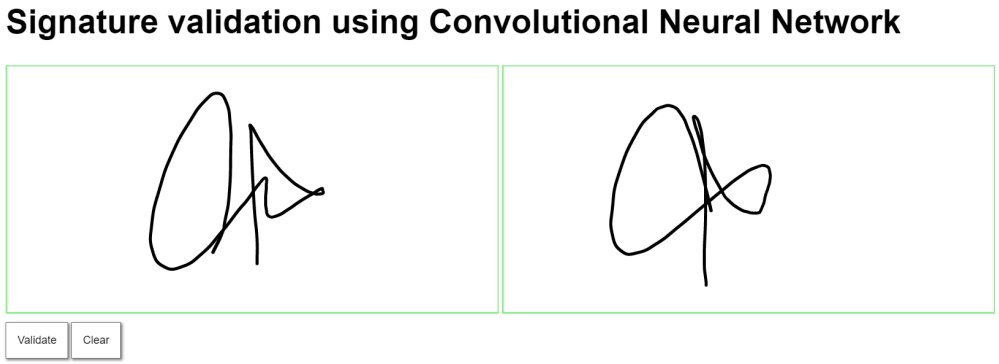

Figura 5.2: Ejemplo de dos firmas validadas (pertenecen al mismo usuario)

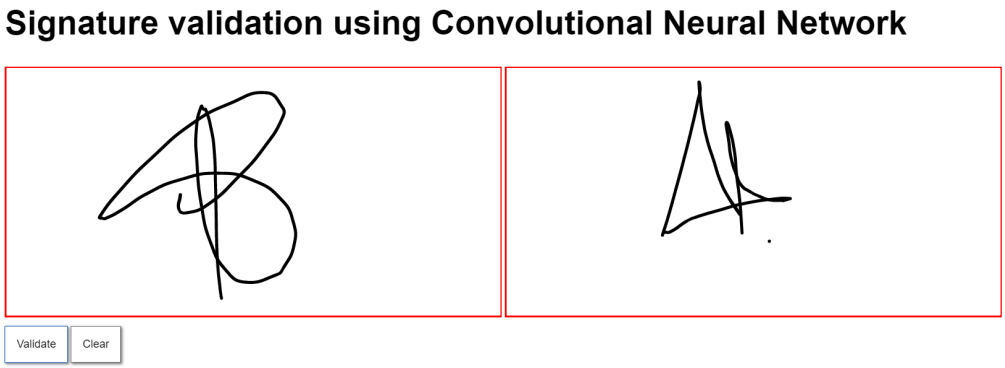

Figura 5.3: Ejemplo de dos firmas no validadas (no pertenecen al mismo usuario)

5.2. Back-end

Esta es la parte donde se encuentra la inteligencia artificial y toda la lógica del sistema y se compone de dos programas principales: 'signature_compare.py' (apéndice B) y 'app.py' (apéndice C).

El primero es el programa donde haremos, para el par de firmas que se encuentren en una determinada carpeta, el proceso de visión por computador explicado. Lo primero que haremos será el pre-procesado de cada una de las imágenes. A continuación haremos la extracción de características y para acabar, el conteo de las mismas en función de los rangos que hemos determinado. Esto nos creará un vector con las características que pertenecen a cada rango, que será la variable de entrada que enviaremos al modelo que hemos creado. Este último nos devolverá el resultado de la predicción.

El segundo programa es el que se encarga de exponer este servicio en una API REST. Esto lo hacemos para que este servicio esté disponible desde cualquier sitio de internet. Para ello hemos usado la librería Falcon Framework [17] escrita en Python. En este programa declararemos el método HTTP POST que recibe las dos imagenes en el 'fromData', las guarda en la carpeta de procesado de imágenes y a continuación llama al servicio 'signature_compare'. Por último recogerá la respuesta de este servicio y la añadirá a la repuesta del método.

Además de estos dos programas, necesitaremos Waitress [19] y Ngrok [18] para exponer este servicio en internet puesto que el front-end que estamos usando está en un entorno público. Waitress es un servidor WSGI puro de Python y Ngrok es un software multi-plataforma que establece túneles seguros desde un punto final público, como Internet, a un servicio de red que se ejecuta localmente, al tiempo que captura todo el tráfico para una inspección y reproducción detalladas. Ngrok dejará de ser necesario una vez nuestros servicios estén instalados en un entorno público en la nube, pero de momento sólo los tenemos en nuestros entornos locales.

Pare entender cómo se enlaza todo, veamos todos los pasos que necesitamos hacer para levantar nuestros servicios:

1- Levantar el entorno de Anaconda que contiene todas las dependencias instaladas (apéndice A) para la ejecución de los servicios. En nuestro caso este entorno se llama tfm.

2- Exponer en alguno de nuestros puertos (en nuestro caso es el 8080) mediante Waitress la API implementada con Falcon:

waitress-serve -port=8080 app:api

3- Exponer con Ngrok el puerto 8080 en una URL pública de internet:

ngrok http 8080

Como hemos podido ver en el front-end, la URL de Ngrok es la que se utiliza para hacer la llamada al servicio.

Capítulo 6

Conclusiones y trabajo futuro

Consideramos que los resultados obtenidos en este trabajo son muy buenos, siempre y cuando se tengan en cuenta los objetivos y limitaciones del mismo. El campo de investigación en el que nos encontramos (reconocimiento de imágenes mediante visión por computador) es tremendamente amplio y se encuentra actualmente en pleno crecimiento, aún con muchos retos y barreras que seguir superando.

Para un futuro trabajo, planteamos las siguientes líneas de investigación:

- **Investigación de nuevos métodos para el pre-procesado de imágenes.**
  Dentro de este campo hay mucho por investigar. Las firmas que estamos testeando con nuestro sistema prototipo son generadas mediante una línea de igual grosor en un fondo totalmente blanco, sin irregularidades. Si la firma es generada en otro tipo de escenario (por ejemplo desde un documento escaneado), hará falta más esfuerzo en este paso para conseguir obtener la máxima información de la firma. Firmas generadas por diferentes bolígrafos de diferente grosor, bolígrafos que están gastados y no nos generan un trazado uniforme serían casos que se nos pueden presentar en este tipo de situaciones y que deberíamos solucionar aplicando nuevos filtros en el pre-procesado de imágenes.

- **Propuesta de nuevas arquitecturas CNN para la extracción de características.**
  La arquitectura que hemos usado para la extracción de características de firmas es una AlexNet. Tal como hemos explicado, esta fue la arquitectura que ganó el

concurso 'ImageNet Large Scale Visual Recognition Challenge (ILSVRC)' en el año 2012. ¿Qué pasaría si probáramos las arquitecturas que ganaron en años posteriores, como podría ser la GoogleNet o la ResNet?

- **Nuevos métodos de clasificación para resolver el problema de clasificación planteado.** Una vez el trabajo haya sido avanzado por alguna línea de investigación, se puede llegar a plantear un cambio en el clasificador que nos permita conseguir mejores resultados. ¿Hay algún modelo que funcione mejor que el random forest?

- **Introducción de firmas falsificadas en el modelo.** Todas las firmas que hemos tratado en el modelo son firmas genuinas, es decir que han sido hechas por el propio usuario. Si quisiéramos adentrarnos en la detección de fraudes, o firmas falsificadas, se nos plantearía un problema diferente; además de si son del mismo usuario o no, nuestro modelo predictivo debería intentar predecir si esa firma en concreto ha sido falsificada. Para ello, necesitaremos una base de datos con firmas falsificadas, o investigar en nuevos métodos que nos permita reconocer este tipo de firmas. Además, en este caso sería muy interesante ver cómo nos podría ayudar recoger información dinámica de la firma (dirección del trazado, puntos de apoyo del bolígrafo...).

Personalmente, creo que realizar este trabajo me ha proporcionado muchas herramientas para poder adentrarme con más seguridad en el campo de la inteligencia artificial y visión por computador. He podido profundizar y ligar muchos de los conceptos impartidos en el máster y hacerme una idea más completa de lo que significa un proyecto que resuelve una problemática dada de este tipo. Además, he podido reforzar mis habilidades técnicas: he aprendido a trabajar con Jupyter, Anaconda y Python, muy bien valorados en la actualidad en el campo de investigación científica, y he aprendido nociones básicas de ReactJS, muy utilizado en los proyectos donde actualmente colaboro como directora de proyectos. Estoy convencida de que este trabajo marcará un antes y un después en mi carrera profesional.

Bibliografía

[1] Luiz G.Hafemann, Robert Sabourin Luiz & S. Oliveira *Learning features for offline handwritten signature verification using deep convolutional neural networks.* Pattern Recognition Volume 70, October 2017, Pages 163-176.

[2] Donato Impedovo & Giuseppe Pirlo *Automatic Signature Verification: The State of the Art* IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) (Volume: 38 , Issue: 5 , Sept. 2008)

[3] Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. (* = equal contribution) *ImageNet Large Scale Visual Recognition Challenge.* IJCV, 2015.

[4] Muhammad Imran Malik *ICDAR 2009 Signature Verification Competition (SigComp2009)* `http://tc11.cvc.uab.es/datasets/SigComp2009_1`

[5] Stuart J. Russell & Peter Norvig *Artificial Intelligence A Modern Approach* Pearson Education, Limited ISBN 978-93-325-4351-5.

[6] E. R. Davies *Computer Vision: Principles, Algorithms, Applications, Learning* Academic Press; Edición: 5 (14 de noviembre de 2017) ISBN-13 978-0128092842.

[7] Ethem Alpaydin *Machine Learning: The New AI* MIT Press Essential Knowledge series ISBN 0262529513.

[8] Giuseppe Bonaccorso *Machine Learning Algorithms: A reference guide to popular algorithms for data science and machine learning* Packt Publishing (July 24, 2017). ISBN-10 1785889621.

[9] Arquitecturas de aprendizaje profundo. `https://www.ibm.com/developerworks/ssa/library/cc-machine-learning-deep-learning-architectures/index.html`

[10] CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more. https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

[11] Essential Classification Algorithms Explained. https://www.kaggle.com/anniepyim/essential-classification-algorithms-explained

[12] Convolutional neural networks. https://developer.ibm.com/articles/cc-convolutional-neural-network-vision-recognition/?mhq=pooling%20convolution

[13] Computer Vision & Image Processing. http://www.isikdogan.com/blog/computer-vision-image-processing.html

[14] Various ways to evaluate a machine learning model's performance. https://towardsdatascience.com/various-ways-to-evaluate-a-machine-learning-models-performance-230449055f15

[15] Understanding Confusion Matrix. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

[16] Signature Pad. https://github.com/szimek/signature_pad

[17] Falcon Framework. https://falconframework.org

[18] Ngrok. https://ngrok.com

[19] Waitress https://docs.pylonsproject.org/projects/waitress

Apéndice

Apéndice A

Dependencias instaladas en el entorno de Anaconda

- alabaster==0.7.11
- appdirs==1.4.3
- asn1crypto==0.24.0
- astroid==2.0.4
- attrs==18.2.0
- Automat==0.7.0
- Babel==2.6.0
- backcall==0.1.0
- backports-abc==0.5
- backports.shutil-get-terminal-size==1.0.0
- backports.shutil-which==3.5.1
- backports.weakref==1.0rc1
- bleach==2.1.4
- certifi==2018.8.24
- cffi==1.11.5
- chardet==3.0.4
- cloudpickle==0.7.0
- colorama==0.3.9
- constantly==15.1.0

- cryptography==2.3.1
- cycler==0.10.0
- Cython==0.29.7
- decorator==4.3.0
- dj-database-url==0.5.0
- Django==2.2.1
- django-heroku==0.3.1
- docutils==0.14
- entrypoints==0.2.3
- falcon==2.0.0
- falcon-multipart==0.2.0
- futures==3.0.3
- gunicorn==19.9.0
- html5lib==1.0.1
- httpie==1.0.2
- hyperlink==17.3.1
- idna==2.7
- imagesize==1.1.0
- incremental==17.5.0
- ipykernel==4.10.0
- ipython==6.5.0
- ipython-genutils==0.2.0
- ipywidgets==7.4.1
- isort==4.3.4
- jedi==0.12.1
- Jinja2==2.10
- jsonschema==2.6.0
- jupyter==1.0.0
- jupyter-client==5.2.3
- jupyter-console==5.2.0
- jupyter-core==4.4.0
- keyring==13.2.1
- kiwisolver==1.0.1
- Lasagne==0.2.dev1
- lazy-object-proxy==1.3.1

- mako==1.0.7
- Markdown==2.2.0
- MarkupSafe==1.0
- matplotlib==3.0.2
- mccabe==0.6.1
- mistune==0.8.3
- mkl-fft==1.0.0
- msgpack-python==0.5.6
- nbconvert==5.3.1
- nbformat==4.4.0
- nose==1.3.7
- notebook==5.6.0
- numpy==1.11.3
- numpydoc==0.8.0
- opencv-python==4.0.0.21
- packaging==17.1
- pandas==0.23.4
- pandocfilters==1.4.2
- parso==0.3.1
- pathlib2==2.3.2
- pickleshare==0.7.4
- Pillow==3.0.0
- prometheus-client==0.3.1
- prompt-toolkit==1.0.15
- protobuf==3.6.1
- psutil==5.4.7
- psycopg2==2.8.2
- pyasn1==0.4.5
- pyasn1-modules==0.2.2
- pycodestyle==2.4.0
- pycparser==2.19
- pydotplus==2.0.2
- pyflakes==2.0.0
- Pygments==2.2.0
- pygpu==0.7.6

- PyHamcrest==1.9.0
- pylint==2.1.1
- pyOpenSSL==18.0.0
- pyparsing==2.3.1
- PySocks==1.6.8
- python-dateutil==2.7.3
- pytz==2018.5
- pywin32==223
- pywinpty==0.5.4
- pyzmq==17.0.0
- QtAwesome==0.5.6
- qtconsole==4.3.1
- QtPy==1.6.0
- requests==2.19.1
- rope==0.11.0
- scandir==1.5
- scikit-learn==0.18.1
- scipy==1.2.1
- seaborn==0.9.0
- Send2Trash==1.5.0
- service-identity==17.0.0
- simplegeneric==0.8.1
- singledispatch==3.4.0.3
- six==1.11.0
- snowballstemmer==1.2.1
- Sphinx==1.7.9
- sphinxcontrib-websupport==1.1.0
- spyder==3.3.1
- spyder-kernels==0.2.6
- sqlparse==0.3.0
- tensorflow==1.2.0
- terminado==0.8.1
- testpath==0.3.1
- Theano==1.0.2+2.gc449c8699
- tornado==5.1.1

- tqdm==4.31.1
- traitlets==4.3.2
- Twisted==18.7.0
- typed-ast==1.1.0
- typing==3.6.6
- urllib3==1.23
- waitress==1.3.0
- wcwidth==0.1.7
- webencodings==0.5.1
- Werkzeug==0.14.1
- whitenoise==4.1.2
- widgetsnbextension==3.4.1
- win-inet-pton==1.0.1
- win-unicode-console==0.5
- wincertstore==0.2
- wrapt==1.10.11
- zope.interface==4.5.0

Apéndice B

Programa:

signatures_compare.py

```
# Libraries Import
from scipy.misc import imread
from preprocess.normalize import preprocess_signature
import signet
from cnn_model import CNNModel
import numpy as np
import math
import array as arr
from sklearn.externals import joblib
import os

def compare():

    canvas_size = (952, 1360)  # Maximum signature size

    # Load and pre-process the two signatures
    #print("Loading images...")

    user_sig = []
    for filename in os.listdir('process/'):
        image = imread('process/'+filename,flatten=1)
        user_sig.append(image)

    # Load the model
    #print("Loading model...")
    model_weight_path = 'models/signet.pkl'
    model = CNNModel(signet, model_weight_path)

```

```
29      def preprocess_image(img):
30          processed_sig = preprocess_signature(img, canvas_size)
31          return model.get_feature_vector(processed_sig, layer='fc2')
32
33      #print("Preprocessing images...")
34      user1_features = preprocess_image(user_sig[0])[0]
35      user2_features = preprocess_image(user_sig[1])[0]
36
37      #Ranges (deducted from previous analysis)
38      r0 = 0.20
39      r1 = 0.4
40      r2 = 0.9
41      r3 = 2
42      r4 = 5
43
44      r = arr.array('i',[0,0,0,0,0,0])
45
46      #print("Computing features...")
47      for f in range(0,2048):
48          delta = math.exp(abs(float(user1_features[f]) - float(user2_features[f↩
                ])))-1
49
50          if delta<r0:
51              r[0] += 1
52          elif delta<r1:
53              r[1] += 1
54          elif delta<r2:
55              r[2] += 1
56          elif delta<r3:
57              r[3] += 1
58          elif delta<r4:
59              r[4] += 1
60          elif delta>=r4:
61              r[5] += 1
62
63      #print("Predicting...")
64      loaded_model = joblib.load('models/finalized_model.sav')
65      result = loaded_model.predict([r])
66      if result==0:
67          print("Signatures do not belong to the same user")
68      else:
69          print("Signatures belong to the same user")
70
71      return result
```

Apéndice C

Programa: app.py

```
import falcon
from falcon_multipart.middleware import MultipartMiddleware
from falcon.http_status import HTTPStatus
import os
import signatures_compare as sc

class Resource(object):

    _CHUNK_SIZE_BYTES = 4096

    # The resource object must now be initialized with a path used during POST
    def __init__(self, storage_path):
        self._storage_path = storage_path

    def on_get(self, req, resp):
        resp.set_header('Access-Control-Allow-Origin', '*')
        resp.set_header('Access-Control-Allow-Methods', '*')
        resp.set_header('Access-Control-Allow-Headers', '*')
        resp.set_header('Access-Control-Max-Age', 1728000)  # 20 days

        resp.body = '{"message": '+str(sc.compare())+'}'
        resp.status = falcon.HTTP_200

    def on_post(self, req, resp):

        resp.set_header('Access-Control-Allow-Origin', '*')
        resp.set_header('Access-Control-Allow-Methods', '*')
        resp.set_header('Access-Control-Allow-Headers', '*')
        resp.set_header('Access-Control-Max-Age', 1728000)  # 20 days

        #Signature 1
```

```
32          input_file1 = req.get_param('sig1')
33          raw1 = input_file1.file.read()
34          filename1 = "Signature1.png"
35          file_path1 = os.path.join(self._storage_path, filename1)
36
37          # Write to a temporary file to prevent incomplete files from
38          # being used.
39          temp_file_path1 = file_path1 + '~'
40
41          # Write the data to a temporary file
42          with open(temp_file_path1, 'wb') as output_file:
43              output_file.write(raw1)
44
45          # Now that we know the file has been fully saved to disk
46          # move it into place.
47          os.rename(temp_file_path1, file_path1)
48
49          #Signature 2
50          input_file2 = req.get_param('sig2')
51          raw2 = input_file2.file.read()
52          filename2 = "Signature2.png"
53          file_path2 = os.path.join(self._storage_path, filename2)
54          temp_file_path2 = file_path2 + '~'
55          with open(temp_file_path2, 'wb') as output_file:
56              output_file.write(raw2)
57          os.rename(temp_file_path2, file_path2)
58
59          resp.body = '{"message": '+str(sc.compare())+'}'
60
61          mydir = 'process/'
62          filelist = [ f for f in os.listdir(mydir) ]
63          for f in filelist:
64              os.remove(os.path.join(mydir, f))
65
66          resp.status = falcon.HTTP_200
67
68  api = application = falcon.API(middleware=[MultipartMiddleware()])
69
70  sig_compare = Resource(storage_path='process')
71  api.add_route('/sig_compare', sig_compare)
```